Práctica 6. Modos de bajo consumo

Esta práctica se desarrollará el 24 de noviembre.

Objetivos

El objetivo de esta práctica es conocer los diferentes modos de bajo consumo que ofrece el ESP32 y la interfaz que expone ESP-IDF para usarlos.

Trabajaremos los siguientes aspectos: * Paso explícito a *light-sleep* y a *deep-sleep*. * Probar diferentes mecanismos para volver de modos de bajo consumo. * Usar el gestor automático de ahorro energético.

Material de consulta

- Descripción de modos de bajo consumo
- Gestión automática de consumo
- Ejemplo sobre light-sleep
- Ejemplo sobre deep-sleep

Modos de bajos consumo

ESP32 ofrece dos modos de ahorro de consumo energético: *Light-sleep* y *Deep-sleep*.

En el modo *light-sleep* se corta la señal de reloj de los periféricos digitales, la mayor parte de la RAM y la CPU; además, se reduce la tensión de alimentación, pero se mantiene por encima de la necesaria para retener la información. Así, el sistema no puede ejecutar ninguna tarea, pero no perdemos la información almacenada en registros y memoria. Al salir de *light-sleep* los periféricos digitales, RAM y CPU continúan su operación como si nada hubiera pasado.

En el modo *deep-sleep* la CPU, la mayor parte de la RAM y los periféricos cuyo señal de reloj viene de *APB_CLK* se apagan por completo. Las únicas partes del chip que mantienen alimentación son:

- Controlador RTC (Real Time Clock). Para mantener la hora del sistema
- Co-procesador ULP (*ultra-lowpower*).
- RTC fast memory
- RTC slow memory

Hay varias fuentes que permiten despertar al sistema cuando se encuentra en *deep-sleep* o en *ligth-sleep*. Podemos especificar varias fuentes de manera que el sistema despierte ante cualquier de ellas. Para configurarlas, ESP-IDF proporciona llamadas de la forma `esp_sleep_enable_X_wakeup()`; para deshabilitarlas, existe `esp_sleep_disable_wakeup_source()`.

Tras configurar la(s) fuente(s) que nos permitirán salir del modo de bajo consumo, podemos entrar en uno de los dos llamando a `esp_light_sleep_start()` or `esp_deep_sleep_start()`. Tanto Wi-Fi como Bluetooth dejarán de funcionar y deberían apagarse antes de entrar en modos de bajo consumo. Si la conexión Wi-Fi debe mantenerse, será necesario utlizar el gestor automático de ahorro energético.

Gestor automático de consunmo

El *Power Manager* proporcionado por ESP-IDF permite controlar la frecuencia las señales de reloj de la CPU y del bus APB, e incluso pasar a *ligth-sleep* en períodos de inactividad. Asimismo permite el uso y creación de *cerrojos* que, en determinadas fases de nuestra aplicación, limitan las acciones del gestor de consumo por determinadas necesidades (mantener una velocidad de APB, mantener interrupciones activas...).

El gestor automático se puede habilitar en tiempo de compilación (a través de *menuconfig*) usando la opción `CONFIG_PM_ENABLE`. Habilitar esta opción aumenta la latencia del tratamiento de interrupciones y disminuye la precisión del reloj del sistema (utilizado para mantener la hora, *timers*...).

Además de hablitar la opción en tiempo de compilación, es necesario configurar su uso en ejecución llamando a `esp_pm_configure()` que recibe un argumento de tipo `esp_pm_config_esp32_t` que tiene tres campos:

- `max_freq_mhz`: frecuencia de CPU máxima en MHz. Es la frecuencia que se utilizará cuando el algún componente adquiera le cerrojo 'ESP_PM_CPU_FREQ_MAX'. Lo habitual es que se deje a la frecuencia por defecto (240MHZ).
- `min_freq_mhz`: frecuencia mínima en MHz. Es la frecuencia que se usará si se adquiere el cerrojo `ESP_PM_APB_FREQ_MAX`.
- `light_sleep_enable`: indica si el sistema debería pasar automáticamente al modo *light-sleep* en períodos de inactividad (con ningún cerrojo adquirido por ningún componente). Es un valor `true` o `false`.

Para poder habilitar la tercera opción (`light_sleep_enable`), debemos habilitar la funcionalidad *Tickless Idle* en *Menuconfig* con la opción `CONFIG_FREERTOS_USE_TICKLESS_IDLE`. En caso contrario, `esp_pm_configure()` devolverá el error *ESP_ERR_NOT_SUPPORTED* si tratamos de habiltar la opción `light_sleep_enable`.

Si todo está configurado correctamente y no hay ningún cerrojo adquirido, el sistema podrá pasar al modo *light-sleep* tras un período de inactividad. El sistema permanecerá en dicho modo en función de eventos como:

- Tareas de FreeRTOS bloqueadas por algún *timeout* finito (como `vTaskDelay()`).
- *Timers* registrados con el API de *High resolution timer*

El sistema saldrá de *light-sleep* para tratar el evento más próximo. Si queremos que nuestros *timers* no nos saquen de *light-sleep* los podemos inicializar con la opción `skip_unhandled_events`.

Librería NVS

La librería *Non-volatile storage* (NVS) está diseñada para almacenar pares *clave-valor* en memoria flash (soporte no volátil). Resulta muy útil para mantener determinados parámetros de diferentes módulos de nuestra aplicación, que queremos mantener almacenados entre diferentes arranques de nuestro sistema.

Para utilizar la librería es necesario disponer de una partición de tipo NVS en nuestro dispositivo flash. Una partición es una porción del dispositivo de almacenamiento (flash en nuestro caso) a la que daremos una identidad específica. Por ejemplo, una partición contendrá la lista de pares *clave-valor* tal y como los organiza la librería NVS. Otra partición puede contener un sistema de ficheros basado en FAT, para que podamos almacenar información y distribuirla en ficheros y directorios.

En la zona inicial de la flash se ubicará la tabla de particiones que indica qué particiones tendremos en nuestro dipositivo y de qué tamaño y tipo son cada una de ellas.

Ejercicios básicos

Vamos a partir del ejemplo que entra manualmente en *light-sleep*.

Tareas

- Hacer funcionar el ejemplo, permitiendo que volvamos de *light-sleep* únicamente por un *timer* o por GPIO.

Cuestión

- ¿Qué número de GPIO está configurado por defecto para despertar al sistema? ¿Está conectado dicho GPIO a algún elemento de la placa *ESP Devkit-c* que estamos usando? Puedes tratar de responder consultando el esquemático de la placa
- ¿Qué flanco provocará que salgamos de *light-sleep* tras configurar el GPIO con `gpio_wakeup_enable(GPIO_WAKEUP_NUM, GPIO_WAKEUP_LEVEL == 0 ? GPIO_INTR_LOW_LEVEL : GPIO_INTR_HIGH_LEVEL)`?

Tareas

- Incluir un *timer* en el código. La aplicación arrancará, configurará un *timer* para que se ejecute su *callback* cada 0.5 segundos, y se dormirá durante 3 segundos (con `vTaskDelay()`). Tras despertar del *delay*, pasará a *light-sleep* (configuraremos el mecanismo de despertar para que lo haga en 5 segundos, si no usamos el GPIO correspondiente). El *callback* del timer simplemente imprimirá un mensaje que incluirá el valor devuelto por `esp_timer_get_time()`.

Cuestión

- ¿Qué observas en la ejecución de los *timer*?¿Se ejecutan en el instante adecuado? ¿Se pierde alguno?

Tareas

- Modifica el código anterior para que, tras 5 pasos por *ligth-sleep*, pasemos a *deep-sleep*. Incluye código para determinar el motivo por el que hemos despertado de *deep-sleep* y muéstralo por pantalla.

Cuestión

- ¿Qué diferencia se observa al volver de *deep-sleep* respecto a volver de *light-sleep*?

Ejercicio final

Integraremos el control de energía en la aplicación de la práctica 3 (monitorización de temperatura)

Tareas

Completar la aplicación de modo que:

- Se configure el gestor de energía para que entre automáticamente en *light-sleep* cuando sea posible.
- Tras 12 horas de funcionamiento, pasará al modo *deep-sleep* durante otras 12 horas (para pruebas, en lugar de 12 horas probadlo con 1 minuto).
- Compruebe el motivo por el que se produce cada reinincio y lo anote en NVS.
- Escriba en NVS la última medida del sensor tomada.

Documentation built with MkDocs.